STLMC: Robust STL Model Checking of Hybrid Systems using SMT

Geunyeol Yu, Jia Lee, and Kyungmin Bae

Pohang University of Science and Technology, Pohang, Korea

Abstract. We present the STLMC model checker for signal temporal logic (STL) properties of hybrid systems. STLMC can perform STL model checking up to a robustness threshold $\epsilon > 0$ for a wide range of nonlinear hybrid systems with ordinary differential equations. Our tool utilizes the refutation-complete SMT-based model checking algorithm with various SMT solvers by reducing the robust STL model checking problem into the Boolean STL model checking problem. If STLMC does not find a counterexample, the system is guaranteed to be correct up to the given bounds and robustness threshold. We demonstrate the effectiveness of STLMC on a number of hybrid system benchmarks.

1 Introduction

Signal temporal logic (STL) [35] has emerged as a popular property specification formalism for hybrid systems. STL formulas describe linear-time properties of continuous real-valued signals. Because hybrid systems exhibit both discrete and continuous behaviors, STL provides a convenient and expressive way to specify important requirements of hybrid systems. STL has a vast range of applications on hybrid systems, including automotive systems [21, 30], medical systems [40], robotics [27, 45], IoT [10], smart cities [22, 34], smart grids [44], etc.

Due to the infinite-state nature of hybrid systems with continuous dynamics, most techniques and tools for analyzing STL properties focus on monitoring and falsification. These techniques analyze concrete samples of signals obtained by simulating hybrid automata to monitor the system's behavior [16, 18, 29, 36] or find counterexamples [1, 4, 41, 48], often combined with stochastic optimization techniques [31]. To this end, STL monitoring and falsification use quantitative semantics that defines the *robustness degree* to indicate how well the formula is satisfied. However, these methods cannot be used to guarantee correctness.

Recently, several STL model checking techniques have been proposed for hybrid systems [7,33,39]. In particular, the SMT-based bounded model checking algorithms [7,33] are refutation-complete and thus can guarantee correctness up to bounds. These techniques are based on the Boolean semantics of STL instead of quantitative semantics. This is certainly a limitation for hybrid systems as small perturbations of signals can cause the system to violate the properties *verified* by Boolean STL model checking. Moreover, there is no tool with a convenient user interface that implements STL model checking techniques.

This paper presents the STLMC tool for robust STL model checking of hybrid systems. Our tool can verify that, up to bounds, the robustness degree of an STL formula φ is greater than a *robustness threshold* $\epsilon > 0$ for all possible behaviors of the system. We reduce the robust STL model checking problem to Boolean STL model checking using *ϵ-strengthening*, first proposed in [25] for first-order logic and extended to STL. We then apply the refutation-complete bounded model checking algorithm [7, 33] to build the SMT encoding of the resulting Boolean STL model checking problem, which can be solved using SMT solvers.

Apart from the robust STL model checking method, STLMC also implements several techniques to improve the usability and scalability of the tool:

- STLMC implements a generic interface to connect with various SMT solvers, such as Z3 [15], Yices2 [20], dReal [26]. Because dReal can (approximately) deal with nonlinear ordinary differential equations (ODEs), STLMC can also support hybrid systems with nonlinear ODE dynamics.
- STLMC implements parallelized two-step SMT solving to improve scalability. Instead of directly solving the complex encoding with ODEs, we first obtain a *discrete abstraction* without ODEs and find satisfying scenarios. We then check the *discrete refinements* of such scenarios using dReal in parallel.
- STLMC provides a visualization command to draw counterexample signals and robustness degrees. Such graphs intuitively explain why the robustness degree of the formula is greater than a given threshold, and thus greatly help in analyzing counterexamples and debugging hybrid systems.

We demonstrate the effectiveness of the STLMC tool on a number of hybrid system benchmarks—including linear, polynomial, and ODE dynamics—and nontrivial STL properties. The tool is available at `https://stlmc.github.io`.

The rest of this paper is organized as follows. Section 2 explains some background on robust STL model checking of hybrid systems using STLMC. Section 3 presents the STLMC language. Section 4 presents the algorithm and implementation of STLMC. Section 5 presents case studies used to evaluate the effectiveness of STLMC. Section 6 shows the experimental results. Section 7 discusses the related work. Finally, Section 8 presents some concluding remarks.

2 Background

2.1 Hybrid Automata

Hybrid automata are widely used for formalizing cyber-physical systems (CPSs) that exhibit both discrete and continuous behaviors. In a hybrid automaton H, a set of *modes* Q specifies discrete states, and a finite set of real-valued *variables* $X = \{x_1, ..., x_l\}$ specifies continuous states. A state of H is a pair $\langle q, \vec{v}\rangle \in Q \times \mathbb{R}^l$ of mode q and real-valued vector $\vec{v}$. An initial condition $\mathit{init}(\langle q, \vec{v}\rangle)$ defines a set of initial states. An invariant condition $\mathit{inv}(\langle q, \vec{v}\rangle)$ defines a set of valid states. A flow condition $\mathit{flow}(\langle q, \vec{v}\rangle, t, \langle q, \vec{v}_t\rangle)$ defines a *continuous evolution* of variables X from a value $\vec{v}$ to $\vec{v}_t$ over duration t in mode q. A jump condition $\mathit{jump}(\langle q, \vec{v}\rangle, \langle q', \vec{v'}\rangle)$ defines a discrete transition from state $\langle q, \vec{v}\rangle$ to $\langle q', \vec{v'}\rangle$.

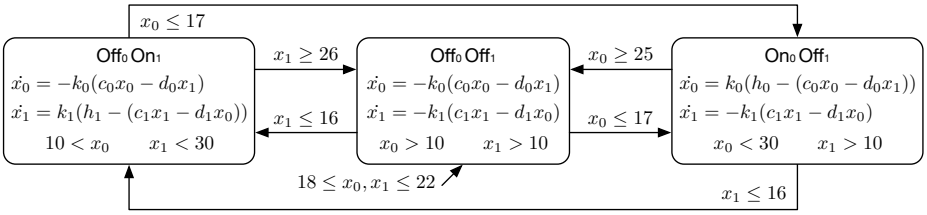

Fig. 1: A hybrid automaton for the networked thermostats.

Definition 1. *A hybrid automaton is a tuple* $H = (Q, X, init, inv, jump, flow)$.

A *signal* σ represents a continuous execution of a hybrid automaton H, given by a function $[0, \tau) \to Q \times \mathbb{R}^l$ with a time bound $\tau > 0$. A signal σ is called *bounded* if $\tau < \infty$. A signal σ is called a *trajectory* of a hybrid automaton H, written $\sigma \in H$, if σ describes a valid behavior of H.

Definition 2. *For a hybrid automaton* H*, a signal* $\sigma : [0, \tau) \to Q \times \mathbb{R}^l$ *is a trajectory of* H*, written* $\sigma \in H$*, if there exist sequences of modes* $q_1, q_2, q_3, ...$ *and of times* $0 = t_0 < t_1 < ... < \tau$ *such that:*

- *the initial condition holds at time* t_0*: i.e.,* $init(\sigma(t_0))$ *holds;*
- *for* $i \geq 1$*, the values of* X *change from* $\sigma(t_{i-1})$ *for time* $t_i - t_{i-1}$ *by the flow condition, satisfying the invariant condition: i.e., for any* $t \in [t_{i-1}, t_i)$*:*

$$flow(\sigma(t_{i-1}), t - t_{i-1}, \sigma(t)) \quad and \quad inv(\sigma(t))$$

- *for* $i \geq 1$*, a discrete jump happens at time* t_i*: i.e., for some* $s_i \in Q \times \mathbb{R}^l$*:*

$$flow(\sigma(t_{i-1}), t_i - t_{i-1}, s_i) \quad and \quad jump(s_i, \sigma(t_i))$$

Example 1. There are two rooms connected by an open door. The temperature x_i of each room $i \in \{0, 1\}$ is controlled by each thermostat, depending on the heater's mode $q_i \in \{\mathsf{On}, \mathsf{Off}\}$ and the other room's temperature. The continuous dynamics of x_i can be given as ODEs as follows [5, 28]:

$$\dot{x_i} = \begin{cases} K_i(h_i - (c_ix_i - d_ix_{1-i})) & (\mathsf{On}) \\ -K_i(c_ix_i - d_ix_{1-i}) & (\mathsf{Off}), \end{cases}$$

where K_i, h_i, c_i, d_i are constants depending on the size of the room, the heater's power, and the size of the door. Figure 1 shows a hybrid automaton of our thermostat controllers. Initially, both heaters are off and the temperatures are between 18 and 22. The jumps between modes then define a control logic to keep the temperatures within a certain range with only one heater on.

2.2 Signal Temporal Logic

Signal temporal logic (STL) can be used to specify properties of CPS. The syntax of STL over a set of state propositions Π is defined by:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \,\mathbf{U}_I\, \varphi$$

where $p \in \Pi$ denotes state propositions and $I \subseteq \mathbb{R}_{\geq 0}$ denotes time intervals of nonnegative real numbers.Examples of state propositions include relational expressions of the form $f(\vec{x}) \geq 0$ with a real-valued function $f : \mathbb{R}^l \to \mathbb{R}$. Other common boolean and temporal operators can be derived by equivalences: e.g.,

$$\varphi \vee \varphi' \equiv \neg(\neg\varphi \wedge \neg\varphi'), \quad \Diamond_I \varphi \equiv \top \,\mathbf{U}_I\, \varphi, \quad \varphi \,\mathbf{R}_I\, \varphi' \equiv \neg((\neg\varphi)\,\mathbf{U}_I(\neg\varphi'))$$

Example 2. For Example 1, we consider the following STL formulas:

- $\Diamond_{[0,3]}\,(x_0 \geq 13\ \mathbf{U}_{[0,\infty)}\ x_1 \leq 22)$: Within 3 seconds, the temperature of room_0 is greater than or equal to 13, until the temperature of room_1 is less than or equal to 22
- $\Box_{[2,4]}\,(x_0 - x_1 \geq 4 \ \rightarrow\ \Diamond_{[3,10]}\, x_0 - x_1 \leq -3)$: For time interval $[2, 4]$, if the temperature of room_0 is 4 degrees higher than or equal to the temperature of room_1, the difference is less than or equal to -3 within time interval $[3, 10]$.
- $\Box_{[0,10]}\,(x_0 > 23\ \mathbf{R}_{[0,\infty)}\ x_0 - x_1 \geq 4)$: For 10 seconds, the temperature of room_0 is 4 degrees higher than or equal to the temperature of room_1, until the temperature of room_0 is greater than 23.

The Minkowski sum of two intervals I and J is denoted by $I + J$. E.g., $[a, b] + [c, d] = [a + c, b + d]$. For a singular interval $\{t\}$, the set $\{t\} + I$ is often written as $t + I$. We write $\sup_{t\in I} g(t)$ and $\inf_{t\in I} g(t)$ to denote the least upper bound and the greatest lower bound of the set $\{g(t) \mid t \in I\}$, respectively.

We consider a quantitative semantics of STL based on *robustness degrees* [18]. The semantics of a proposition p is specified as a function $p : Q \times \mathbb{R}^l \to \overline{\mathbb{R}}$, where $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$, that assigns to a state of H the degree to which p is true. For a state proposition of the form $f(\vec{x}) \geq 0$, the robustness degree is the value of $f(\vec{x})$ at a given state. E.g., the robustness degree of proposition $x_0 - x_1 \geq 4$ in Example 2 is the value of $x_0 - x_1 - 4$ at a given state.

The robustness degree of an STL formula can be defined inductively [18]. We explicitly take into account a time bound τ of a signal.

Definition 3. *Given an STL formula φ, a signal $\sigma : [0, \tau) \to \mathbb{R}^l$, and a time $t \in [0, \tau)$, the robustness degree $\rho_\tau(\varphi, \sigma, t) \in \overline{\mathbb{R}}$ is defined inductively by:*

$$\begin{aligned}
\rho_\tau(p, \sigma, t) &= p(\sigma(t)) \\
\rho_\tau(\neg\varphi, \sigma, t) &= -\rho_\tau(\varphi, \sigma, t) \\
\rho_\tau(\varphi_1 \wedge \varphi_2, \sigma, t) &= \min(\rho_\tau(\varphi_1, \sigma, t), \rho_\tau(\varphi_2, \sigma, t)) \\
\rho_\tau(\varphi_1\,\mathbf{U}_I\,\varphi_2, \sigma, t) &= \sup_{t'\in(t+I)\cap[0,\tau)} \min(\rho_\tau(\varphi_2, \sigma, t'), \inf_{t''\in[t,t']} \rho_\tau(\varphi_1, \sigma, t''))
\end{aligned}$$

The robust STL model checking problem is to check whether an STL formula φ always has a greater robustness degree than a given threshold $\epsilon > 0$ for any possible trajectory of a hybrid automaton H.

Definition 4 (Robust STL Model Checking). *For a time bound $\tau > 0$, an STL formula φ is satisfied at time $t \in [0, \tau)$ on a hybrid automaton H with respect to a robustness threshold $\epsilon > 0$ iff for every trajectory $\sigma \in H$, $\rho_\tau(\varphi, \sigma, t) > \epsilon$.*

2.3 STL Boolean Semantic

We also introduce the Boolean semantics of STL, which was originally proposed as the semantics of STL in [35]. We consider the following definition from [33] that explicitly considers a time bound $\tau > 0$.

Definition 5. *Given an STL formula φ, a signal $\sigma : [0, \tau) \to \mathbb{R}^l$, and a time $t \in [0, \tau)$, the satisfaction of φ, denoted by $\sigma, t \models_\tau \varphi$ is inductively defined by:*

$$
\begin{array}{ll}
\sigma, t \models_\tau p & \textit{iff } t < \tau \textit{ and } p(\sigma(t)) = \top \\
\sigma, t \models_\tau \neg\varphi & \textit{iff } \sigma, t \not\models_\tau \varphi \\
\sigma, t \models_\tau \varphi_1 \wedge \varphi_2 & \textit{iff } \sigma, t \models_\tau \varphi_1 \textit{ and } \sigma, t \models_\tau \varphi_2 \\
\sigma, t \models_\tau \varphi_1 \, \mathbf{U}_I \, \varphi_2 & \textit{iff } \exists t' \in (t + I) \cap [0, \tau).\ \sigma, t' \models_\tau \varphi_2,\ \forall t'' \in [t, t'].\ \sigma, t'' \models_\tau \varphi_1
\end{array}
$$

The following theorem shows the obvious relationship between the Boolean semantics and the quantitative semantics. If the robustness degree of φ is strictly positive (resp., negative), then φ is true (resp., false).

Theorem 1. *[18] For an STL formula φ, a signal σ, and a time $t \in [0, \tau)$, $\rho_\tau(\varphi, \sigma, t) > 0$ implies $\sigma, t \models_\tau \varphi$, and $\rho_\tau(\varphi, \sigma, t) < 0$ implies $\sigma, t \not\models_\tau \varphi$.*

3 The STLmc Tool

The STLmc tool can model check STL properties of hybrid automata, given three parameters $\epsilon > 0$ (robustness threshold), $\tau > 0$ (time bound), and $N \in \mathbb{N}$ (discrete bound). STLmc provides an expressive input format to easily specify a wide range of hybrid automata. STLmc also provides a visualization command to give an intuitive description of counterexamples.

3.1 Input Format

The input format of STLmc, inspired by dReach [32], consists of five sections: variable declarations, mode definitions, initial conditions, state propositions, and STL properties. Mode and continuous variables specify discrete and continuous states of hybrid automata. Mode definitions specify flow, jump, and invariant conditions. STL formulas can also include user-defined state propositions.

Variable Declarations. STLMC uses mode and continuous variables to specify discrete and continuous states of a hybrid automaton. *Mode variables* define a set of discrete modes Q, and *continuous variables* define a set of real-valued variables X.[1] We can also declare named constants whose values do not change.

Mode variables are declared with one of three types: `bool` variables with `true` and `false` values, `int` variables with integer values, and `real` variables with real values. E.g., the following declares a `bool` variable `b` and an `int` variable `i`:

```
bool b;    int i;
```

Continuous variables are declared with domain intervals. Domains can be any intervals of real numbers, including open, closed, and half-open intervals, E.g., the following declares two continuous variable `x` and `y`:

```
[0, 50] x;    (-1.1, 1) y;
```

Finally, constants are introduced with the `const` keyword. Constants can have Boolean, integer, or rational values. For example, the following declares a constant `k1` with a rational value 0.015:

```
const k1 = 0.015;
```

Mode Definitions. In STLMC, *mode blocks* define mode, jump, invariant, and flow conditions for a group of modes in a hybrid automaton. Multiple model blocks can be declared, and each mode block consists of four components:

```
{
    mode: ...
    inv:  ...
    flow: ...
    jump: ...
}
```

A `mode` component contains a semicolon-separated set of Boolean conditions over mode variables. The conjunction of these conditions represents a set of modes.[2] E.g., the following represents a set of two modes $\{(true, 1), (true, 2)\}$, provided there are two mode variables `b` (of `bool` type) and `i` (of `int` type):

```
b = true;    i > 0;    i < 3;
```

An `inv` component contains a semicolon-separated set of Boolean formulas over continuous variables. The conjunction of these conditions then represents the invariant condition for each mode in the mode block. For example, the following declares an invariant condition for two continuous variables `x` and `y`:

[1] An assignment to mode and continuous variables corresponds to a single state.

[2] We assume that the mode conditions of different mode blocks cannot be satisfied at the same time, which can be automatically detected by our tool.

```
x < 30;    y > - 0.5;
```

A `flow` component contains either a system of ordinary differential equations (ODEs) or a closed-form solution of ODEs. In STLMC, a system of ODEs over continuous variables $x_1, \ldots, x_n$ is written as a semicolon-separated equation of the following form, where e_i denotes an expression over $x_1, \ldots, x_n$:

```
d/dt[x1] = e1(x1, ..., xn) ;
...
d/dt[xn] = en(x1, ..., xn) ;
```

A closed-form solution of ODEs is written as a set of continuous functions, parameterized by a time variable `t` and the initial values $x_1(0), ..., x_n(0)$:

```
x1(t) = e1(t, x1(0),...,xn(0)) ;
...
xn(t) = en(t, x1(0),...,xn(0)) ;
```

A `jump` component contains a set of jump conditions *guard* `=>` *reset*, where *guard* and *reset* are Boolean conditions over mode and continuous variables. We use "primed" variables to denote states after jumps have occurred. E.g., the following defines a jump with four variables `b`, `i`, `x`, and `y` (declared above):

```
(and (i = 1) (x > 10)) => (and (b' = false) (i = 3) (x' = x) (y' = 0));
```

Initial Conditions. In the `init` section, an initial condition is declared as a set of Boolean formulas over mode and continuous variables. Similarly, the conjunction of these conditions represents a set of initial modes. E.g., the following shows an initial condition with variables `b`, `i`, `x`, and `y`:

```
init: (and (not b) (i = 0) (19.9 <= x) (y = 0));
```

STL Properties. In the `goal` section, STL properties are declared with labels. State propositions are arithmetic and relational expressions over mode and continuous variables, and labels can be omitted. For example, the following declares the first STL formula in Example 2 with label `f2`:

```
f2: [][2, 4]((x0 - x1 >= 4) -> <>[3, 10] (x0 - x1 <= -3));
```

To make it easy to write repeated propositions, "named" state propositions can be declared in the `proposition` section. For example, the above STL formula can be rewritten using two propositions `p1` and `p2` as follows:

```
proposition:
[p1]: x0 - x1 >= 4;
[p2]: x0 - x1 <= -3;
```

```
goal:
[f2]: [][2, 4](p1 -> <>[3, 10] p2);
```

```
const k0 = 0.015;        const k1 = 0.045;
const h0 = 100;          const h1 = 200;
const c0 = 0.98;         const c1 = 0.97;
const d0 = 0.01;         const d1 = 0.03;

int on0;                 int on1;
[10, 35] x0;             [10, 35] x1;

{ mode: on0 = 0;          on1 = 1;
  inv:  10 < x0; x1 < 30;
  flow: d/dt[x0] = - k0 * (c0 * x0 - d0 * x1);
        d/dt[x1] = k1 * (h1 - (c1 * x1 - d1 * x0));
  jump: x0 <= 17 => (and (on0' = 1) (on1' = 0)
                         (x0' = x0) (x1' = x1));
        x1 >= 26 => (and (on1' = 0) (on0' = on0)
                         (x0' = x0) (x1' = x1));
}
{ mode: on0 = 1;          on1 = 0;
  inv:  x0 < 30; x1 > 10;
  flow: d/dt[x0] = k0 * (h0 - (c0 * x0 - d0 * x1));
        d/dt[x1] = - k1 * (c1 * x1 - d1 * x0);
  jump: x1 <= 16 => (and (on0' = 0) (on1' = 1)
                         (x0' = x0) (x1' = x1));
    x0 >= 25 => (and (on0' = 0) (on1' = on1)
                     (x0' = x0) (x1' = x1));
}
{ mode: on0 = 0;          on1 = 0;
  inv:  x0 > 10; x1 > 10;
  flow: d/dt[x0] = - k0 * (c0 * x0 - d0 * x1);
        d/dt[x1] = - k1 * (c1 * x1 - d1 * x0);
  jump:
    x0 <= 17 => (and (on0' = 1) (on1' = on1)
                     (x0' = x0) (x1' = x1));
    x1 <= 16 => (and (on1' = 1) (on0' = on0)
                     (x0' = x0) (x1' = x1));
}

init: on0 = 0;  18 <= x0;  x0 <= 22;
       on1 = 0;  18 <= x1;  x1 <= 22;

proposition:
  [p1]: x0 - x1 >= 4;    [p2]: x0 - x1 <= -3;

goal:
  [f1]: <>[0, 3](x0 >= 13 U[0, inf) x1 <= 22);
  [f2]: [][2, 4](p1 -> <>[3, 10] p2);
```

Fig. 2: An input model example

Figure 2 shows the input model of the hybrid automaton described in the running example above. Constants are introduced with the `const` keyword. Two mode variables `on0` and `on1` denote the heaters' modes. Continuous variables `x0` and `x1` are declared with domain intervals. There are three "mode blocks" that specify the three modes in Fig. 1 and their invariant, flow, jump conditions.

In mode blocks, a `mode` component includes a set of logic formulas over mode variables. An `inv` component contains a set of logic formulas over continuous variables. A `flow` component can include ODEs over continuous variables. A `jump` component contains a set of jump conditions *guard* `=>` *reset*, where *guard* and *reset* are logic formulas over mode and continuous variables, and "primed" variables denote states after the jump has occurred.

STL properties are declared in the `goal` section, and "named" propositions are declared in the `proposition` section. State propositions can include arithmetic and relational expressions over mode and continuous variables. For example, in Fig 2, the STL formula `f1` contains two state propositions $x_0 \geq 13$ and $x_1 \leq 22$, and the formula `f2` contains the user-defined propositions `p1` and `p2`.

Option	Explanation	Default
`-bound <`N`>`	a discrete bound N	-
`-time-bound <`τ`>`	a time bound τ	-
`-time-horizon <`T`>`	a mode duration bound T	τ
`-threshold <`ϵ`>`	a robustness threshold ϵ	0.01
`-goal`	the list of STL goals to be analyzed	all
`-two-step`	use the two-phase optimization	disabled
`-parallel`	parallelize the two-phase optimization	disabled
`-visualize`	generate extra visualization data	disabled
`-solver <Solver>`	an SMT solver to be used (`z3`/`yices`/`dreal`)	`z3`
`-precision <`δ`>`	a precision parameter for `dreal`	0.001

Table 1: The command line options of STLmc.

3.2 Command Line Options

The STLmc tool provides a command-line interface with various command line options, summarized in Table 1. A discrete bound N limits the number of mode changes and the number of *variable points*—at which the truth value of some STL subformula changes—in trajectories. A time horizon T limits the maximum time duration of single modes in trajectories. The options `-two-step` and `-parallel` enable the (parallelized) two-phase optimization in Sec. 4.3. When the option `-visualize` is set, extra data for visualization is generated.

We can choose different SMT solvers using the `-solver` option. The STLmc tool currently supports three SMT solvers: Z3 [15], Yices2 [20] and dReal [26]. The underlying solver is chosen depending on the flow conditions of a hybrid automaton. Z3 and Yices2 can deal with linear and polynomial functions, and dReal can deal with nonlinear ODEs—which is undecidable in general for hybrid automata—approximately up to a given precision $\delta > 0$.

Example 3. Consider the input model in Fig. 2 (`therm.model`). The following command found a counterexample of the formula `f2` at bound 2 with respect to robustness threshold $\epsilon = 2$ in 8 seconds using dReal.

```
$./stlmc ./therm.model -bound 5 -time-bound 30 -threshold 2 \
         -goal f2 -solver dreal -two-step -parallel -visualize
goal: []_[2.0,4.0] (p1 -> (<>_[3.0,10.0] p2))
result: counterexample found (bound 2)
running time: 7.46335 seconds
```

The following command verifies the formula `f1` up to bounds $N = 5$ and $\tau = 30$ with respect to robustness threshold $\epsilon = 1$ in 817 seconds using dReal.

```
$./stlmc ./therm.model -bound 5 -time-bound 30 -threshold 1 \
         -goal f1 -solver dreal -two-step -parallel
goal : (<>_[0.0,3.0] ((x0 >= 13) U_[0.0,inf) (x1 <= 22)))
result : True
```

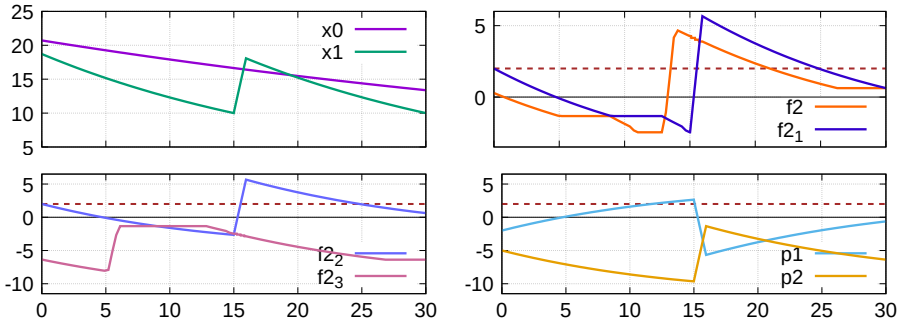

Fig. 3: Visualization of a counterexample (horizontal dotted lines denote $\epsilon = 2$).

```
running time 816.99192 seconds
```

3.3 Visualization

The STLMC tool provides a script to visualize counterexamples for robust STL model checking. The visualization command can generate HTML files or PDF images that contain graphs to represent counterexample trajectories of a hybrid automaton or robustness degrees for each subformula of an STL formula φ.

Example 4. Consider the model checking command for `f2` in Example 3, where `-visualize` is enabled. The following command generate a PDF image.

```
./stlmc-vis ./therm.model -output pdf
```

Figure 3 shows the (slightly adapted) visualization graphs generated for $\texttt{f2} = \square_{[2,4]}(x_0 - x_1 \geq 4 \rightarrow \Diamond_{[3,10]}(x_0 - x_1 \leq -3))$ with the subformulas:

$$\texttt{f2}_1 = x_0 - x_1 \geq 4 \rightarrow \Diamond_{[3,10]}(x_0 - x_1 \leq -3) \qquad \texttt{f2}_2 = \neg(x_0 - x_1 \geq 4)$$
$$\texttt{f2}_3 = \Diamond_{[3,10]}(x_0 - x_1 \leq -3) \qquad \texttt{p}_1 = x_0 - x_1 \geq 4 \qquad \texttt{p}_2 = x_0 - x_1 \leq -3$$

The robustness degree of `f2` is less than ϵ at time 0, since the robustness degree of $\texttt{f2}_1$ goes below ϵ in the interval $[2, 4]$, which is because both the degrees of $\texttt{f2}_2$ and $\texttt{f2}_3$ are less than ϵ in $[2, 4]$. The robustness degree of $\texttt{f2}_3$ is less than ϵ in $[2, 4]$, since the robustness degree of $\texttt{p}_2$ is less than ϵ in $[5, 14] = [2, 4] + [3, 10]$.

4 Algorithms and Implementation

This section explains the algorithms and implementation of our STLMC tool. Section 4.1 explains how to reduce the robust STL model checking problem to the Boolean STL model checking problem. Section 4.2 summarizes a Boolean STL model checking algorithm, introduced in [7,33]. Section 4.3 explains a two-phase solving optimization to improve the performance for ODEs. Finally, Section 4.4 presents the tool's architecture and implementation.

4.1 Reduction to Boolean STL Model Checking

As usual for model checking, robust STL model checking is equivalent to finding a counterexample. We can easily see that an STL formula φ is not satisfied in a hybrid automata H with respect to a robustness threshold $\epsilon > 0$ iff there is a trajectory that the robustness degree of $\neg\varphi$ is greater than or equal to $-\epsilon$.[3]

Corollary 1. *For a hybrid automaton H, a time $t \in [0, \tau)$, and a robustness threshold $\epsilon > 0$, an STL formula φ is not satisfied at t in H with respect to ϵ iff there exists a trajectory $\sigma \in H$ such that $\rho_\tau(\neg\varphi, \sigma, t) \geq -\epsilon$.*

Our goal is to reduce finding a counterexample of robust STL model checking into finding a counterexample of Boolean STL model checking. Let us first a simple case, say, a state proposition $x \geq 0$. Its robust model checking with respect ϵ is equivalent to finding a counterexample $\sigma \in H$ with $\rho_\tau(-x > 0, \sigma, t) \geq -\epsilon$ by Corollary 1, which is equivalent to $\rho_\tau(-x > -\epsilon, \sigma, t) \geq 0$. Notice that $-x > -\epsilon$ can be obtained by *weakening* $-x \geq 0$ by ϵ.

The notions of ϵ-weakening ϵ-strengthening are first introduced in [25] for first-order formulas. In this paper, we define the notions of ϵ-weakening and ϵ-strengthening for STL formulas as follows.

Definition 6. *The ϵ-weakening $\varphi^{-\epsilon}$ and ϵ-strengthening $\varphi^{+\epsilon}$ of φ are defined as follows: $(p^{-\epsilon})(s) = p(s) - \epsilon$ and $(p^{+\epsilon})(s) = p(s) + \epsilon$ for a state s, and:*

$$(\neg\varphi)^{-\epsilon} \equiv \neg(\varphi^{+\epsilon}) \quad (\varphi_1 \wedge \varphi_2)^{-\epsilon} \equiv \varphi_1^{-\epsilon} \wedge \varphi_2^{-\epsilon} \quad (\varphi_1 \,\mathbf{U}_I\, \varphi_2)^{-\epsilon} \equiv \varphi_1^{-\epsilon} \,\mathbf{U}_I\, \varphi_2^{-\epsilon}$$
$$(\neg\varphi)^{+\epsilon} \equiv \neg(\varphi^{-\epsilon}) \quad (\varphi_1 \wedge \varphi_2)^{+\epsilon} \equiv \varphi_1^{+\epsilon} \wedge \varphi_2^{+\epsilon} \quad (\varphi_1 \,\mathbf{U}_I\, \varphi_2)^{+\epsilon} \equiv \varphi_1^{+\epsilon} \,\mathbf{U}_I\, \varphi_2^{+\epsilon}$$

The following lemma states that the relationship between the satisfaction of an STL formula in robustness STL model chekcing and the satisfactions of $\varphi^{+\epsilon}$ and $\varphi^{-\epsilon}$ in STL model checking.

Lemma 1. *For a signal σ and an STL property φ:*

(1) $\rho_\tau(\varphi, \sigma, t) > \epsilon \;\Rightarrow\; \sigma, t \models_\tau \varphi^{+\epsilon}$
(2) $\rho_\tau(\varphi, \sigma, t) < -\epsilon \;\Rightarrow\; \sigma, t \not\models_\tau \varphi^{-\epsilon}$

Proof. The proof is by structural induction on φ

(1) Consider $\varphi = p$, $\rho_\tau(p, \sigma, t) > \epsilon$ iff $p(\sigma(t)) > \epsilon$ by definition. Then, $\sigma, t \models_\tau p^{+\epsilon}$. Consider $\varphi = \neg\phi$, $\rho_\tau(\neg\phi, \sigma, t) > \epsilon$ iff $\rho_\tau(\phi, \sigma, t) < -\epsilon$ by definition. If $\rho_\tau(\phi, \sigma, t) < -\epsilon$, then $\sigma, t \not\models_\tau \phi^{-\epsilon}$ by Lemma 1.(2). Since $\neg\phi^{-\epsilon}$ is equal to $(\neg\phi)^{+\epsilon}$, $\sigma, t \models_\tau (\neg\phi)^{+\epsilon}$ is also satisfied. Consider $\varphi = \varphi_1 \wedge \varphi_2$, $\rho_\tau(\varphi_1 \wedge \varphi_2, \sigma, t) > \epsilon$ iff $\rho_\tau(\varphi_1, \sigma, t) > \epsilon \wedge \rho_\tau(\varphi_2, \sigma, t) > \epsilon$. Then, $\sigma, t \models_\tau \varphi_1^{+\epsilon} \wedge\ \sigma, t \models_\tau \varphi_2^{+\epsilon}$ by induction hypothesis. Thus, $\sigma, t \models_\tau (\varphi_1 \wedge \varphi_2)^{+\epsilon}$.
Consider $\varphi = \varphi_1 \,\mathbf{U}_I\, \varphi_2$. There is a time point $t' \in (t + I) \cap [0, \tau)$ such that

[3] By definition, φ is satisfied at t in H with respect to ϵ iff for any trajectory $\sigma \in H$, $\rho_\tau(\varphi, \sigma, t) > \epsilon$ holds. Notice that $\neg(\forall\sigma \in H.\ \rho_\tau(\varphi, \sigma, t) > \epsilon)$ iff $\exists\sigma \in H.\ \rho_\tau(\varphi, \sigma, t) \leq \epsilon$ iff $\exists\sigma \in H.\ -\rho_\tau(\varphi, \sigma, t) \geq -\epsilon$ iff $\exists\sigma \in H.\ \rho_\tau(\neg\varphi, \sigma, t) \geq -\epsilon$.

$\min(\rho_\tau(\varphi_2,\sigma,t'), \inf_{t''\in[t,t']} \rho_\tau(\varphi_1,\sigma,t'')) > \epsilon$. Then, $\exists t' \in (t+I)\cap[0,\tau)$, $\rho_\tau(\varphi_2,\sigma,t') > \epsilon \wedge \inf_{t''\in[t,t']} \rho_\tau(\varphi_1,\sigma,t'') > \epsilon$. If $\inf_{t''\in[t,t']} \rho_\tau(\varphi_1,\sigma,t'') > \epsilon$ is satisfied, then $\forall t'' \in [t,t'], \rho_\tau(\varphi_1,\sigma,t'') > \epsilon$.
Thus, $\sigma,t' \models_\tau \varphi_2^{+\epsilon} \wedge\ \forall t'' \in [t,t'], \sigma,t \models_\tau \varphi_1^{+\epsilon}$, by induction hypothesis. Therefore, $\sigma,t \models_\tau (\varphi_1\,\mathbf{U}_I\,\varphi_2)^{+\epsilon}$.

(2) Consider $\varphi = p$, $\rho_\tau(p,\sigma,t) < -\epsilon$ iff $p(\sigma(t)) < -\epsilon$ by definition. Then, $\sigma,t \not\models_\tau p^{-\epsilon}$. Consider $\varphi = \neg\phi$, $\rho_\tau(\neg\phi,\sigma,t) < -\epsilon$ iff $\rho_\tau(\phi,\sigma,t) > \epsilon$ by definition. If $\rho_\tau(\phi,\sigma,t) > \epsilon$, then $\sigma,t \models_\tau \phi^{+\epsilon}$ by Lemma 1.(1). Since $\neg\neg(\phi^{+\epsilon})$ is equal to $(\phi^{+\epsilon})$ and $\neg(\phi^{+\epsilon})$ is equal to $(\neg\phi)^{-\epsilon}$, $\sigma,t \not\models_\tau (\neg\phi)^{-\epsilon}$ is also satisfied. Consider $\varphi = \varphi_1 \wedge \varphi_2$, $\rho_\tau(\varphi_1\wedge\varphi_2,\sigma,t) < -\epsilon$ iff $\rho_\tau(\varphi_1,\sigma,t) < -\epsilon \vee \rho_\tau(\varphi_2,\sigma,t) < -\epsilon$. Then, $\sigma,t \not\models_\tau \varphi_1^{-\epsilon} \vee\ \sigma,t \not\models_\tau \varphi_2^{-\epsilon}$ by induction hypothesis. Thus, $\sigma,t \not\models_\tau (\varphi_1 \wedge \varphi_2)^{-\epsilon}$.
Consider $\varphi = \varphi_1\,\mathbf{U}_I\,\varphi_2$. For all time points $t' \in (t+I)\cap[0,\tau)$, $\min(\rho_\tau(\varphi_2,\sigma,t'), \inf_{t''\in[t,t']} \rho_\tau(\varphi_1,\sigma,t'')) < -\epsilon$. Then, $\forall t' \in (t+I)\cap[0,\tau)$, $(\rho_\tau(\varphi_2,\sigma,t') < -\epsilon \vee \inf_{t''\in[t,t']} \rho_\tau(\varphi_1,\sigma,t'') < -\epsilon)$. If $\inf_{t''\in[t,t']} \rho_\tau(\varphi_1,\sigma,t'') < -\epsilon$ is satisfied, then $\exists t'' \in [t,t'], \rho_\tau(\varphi_1,\sigma,t'') < -\epsilon$.
Thus, $\forall t' \in (t+I)\cap[0,\tau), (\sigma,t' \not\models_\tau \varphi_2^{-\epsilon} \vee \exists t'' \in [t,t'], \sigma,t \not\models_\tau \varphi_1^{-\epsilon})$, by induction hypothesis. Therefore, $\sigma,t \not\models_\tau (\varphi_1\,\mathbf{U}_I\,\varphi_2)^{-\epsilon}$.

Lemma 2. *For a signal σ and an STL property φ:*

(1) $\rho_\tau(\varphi,\sigma,t) < \epsilon \;\Rightarrow\; \sigma,t \not\models_\tau \varphi^{+\epsilon}$
(2) $\rho_\tau(\varphi,\sigma,t) > -\epsilon \;\Rightarrow\; \sigma,t \models_\tau \varphi^{-\epsilon}$

Proof. The proof is by structural induction on φ

(1) Consider $\varphi = p$, $\rho_\tau(p,\sigma,t) < \epsilon$ iff $p(\sigma(t)) < \epsilon$ by definition. Then, $\sigma,t \not\models_\tau p^{+\epsilon}$. Consider $\varphi = \neg\phi$, $\rho_\tau(\neg\phi,\sigma,t) < \epsilon$ iff $\rho_\tau(\phi,\sigma,t) > -\epsilon$ by definition. If $\rho_\tau(\phi,\sigma,t) > -\epsilon$ is satisfied, then $\sigma,t \models_\tau \phi^{-\epsilon}$ by Lemma 2. (2). Since $\phi^{-\epsilon}$ is equal to $\neg\neg(\phi^{-\epsilon})$ and $\neg(\phi^{-\epsilon})$ is equal to $(\neg\phi)^{+\epsilon}$, $\sigma,t \not\models_\tau (\neg\phi)^{+\epsilon}$ is also satisfied. Consider $\varphi = \varphi_1\wedge\varphi_2$, $\rho_\tau(\varphi_1\wedge\varphi_2,\sigma,t) < \epsilon$ iff $\rho_\tau(\varphi_1,\sigma,t) < \epsilon \vee \rho_\tau(\varphi_2,\sigma,t) < \epsilon$. Then, $\sigma,t \not\models_\tau \varphi_1^{+\epsilon} \vee \sigma,t \not\models_\tau \varphi_2^{+\epsilon}$ by induction hypothesis. Thus, $\sigma,t \not\models_\tau (\varphi_1 \wedge \varphi_2)^{+\epsilon}$.
Consider $\varphi = \varphi_1\,\mathbf{U}_I\,\varphi_2$. For all time points $t' \in (t+I)\cap[0,\tau)$, $\min(\rho_\tau(\varphi_2,\sigma,t'), \inf_{t''\in[t,t']} \rho_\tau(\varphi_1,\sigma,t'')) < \epsilon$. Then, $\forall t' \in (t+I)\cap[0,\tau)$, $(\rho_\tau(\varphi_2,\sigma,t') < \epsilon \vee \inf_{t''\in[t,t']} \rho_\tau(\varphi_1,\sigma,t'') < \epsilon)$. $\inf_{t''\in[t,t']} \rho_\tau(\varphi_1,\sigma,t'') < \epsilon$, iff $\exists t'' \in [t,t'], \rho_\tau(\varphi_1,\sigma,t'') < \epsilon$. Thus, $\forall t' \in (t+I)\cap[0,\tau)$, $(\sigma,t' \not\models_\tau \varphi_2^{+\epsilon} \vee \exists t'' \in [t,t'],\ \sigma,t'' \not\models_\tau \varphi_1^{+\epsilon})$, by induction hypothesis.
Therefore, $\sigma,t \not\models_\tau (\varphi_1\,\mathbf{U}_I\,\varphi_2)^{+\epsilon}$.

(2) Consider $\varphi = p$, $\rho_\tau(p,\sigma,t) > -\epsilon$ iff $p(\sigma(t)) > -\epsilon$ by definition. Then, $\sigma,t \models_\tau p^{-\epsilon}$. Consider $\varphi = \neg\phi$, $\rho_\tau(\neg\phi,\sigma,t) > -\epsilon$ iff $\rho_\tau(\phi,\sigma,t) < \epsilon$ by definition. If $\rho_\tau(\phi,\sigma,t) < \epsilon$ is satisfied, then $\sigma,t \not\models_\tau \phi^{\epsilon}$ by Lemma 2. (1). Since $\neg\phi^{-\epsilon}$ is equal to $(\neg\phi)^{+\epsilon}$, $\sigma,t \models_\tau (\neg\phi)^{-\epsilon}$ is also satisfied. Consider $\varphi = \varphi_1 \wedge \varphi_2$, $\rho_\tau(\varphi_1 \wedge \varphi_2,\sigma,t) > -\epsilon$ iff $\rho_\tau(\varphi_1,\sigma,t) > -\epsilon \wedge \rho_\tau(\varphi_2,\sigma,t) > -\epsilon$.
Then, $\sigma,t \models_\tau \varphi_1^{-\epsilon} \wedge \sigma,t \models_\tau \varphi_2^{-\epsilon}$ by induction hypothesis. Thus, $\sigma,t \models_\tau (\varphi_1\wedge\varphi_2)^{-\epsilon}$.
Consider $\varphi = \varphi_1\,\mathbf{U}_I\,\varphi_2$. There is a time point $t' \in (t+I)\cap[0,\tau)$ such that

$\min(\rho_\tau(\varphi_2, \sigma, t'), \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t'')) > -\epsilon$.
Then, $\exists t' \in t + I \cap [0, \tau)$, $(\rho_\tau(\varphi_2, \sigma, t') > -\epsilon \wedge \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t'') > -\epsilon)$.
$\inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t'') > -\epsilon$, iff $\forall t'' \in [t, t'], \rho_\tau(\varphi_1, \sigma, t'') > -\epsilon$.
Thus, $\exists t' \in t + I \cap [0, \tau)$, $(\sigma, t' \models_\tau \varphi_2^{-\epsilon} \wedge \forall t'' \in [t, t'], \sigma, t \models_\tau \varphi_1^{-\epsilon})$, by induction hypothesis. Therefore, $\sigma, t \models_\tau (\varphi_1 \, \mathbf{U}_I \, \varphi_2)^{-\epsilon}$.

According to above lemmas, we can reduce finding a counterexample for robust STL model checking of an STL formula φ into finding a counterexample for Boolean STL model checking of the ϵ-weakening of $\neg\varphi$, i.e., $(\neg\varphi)^{-\epsilon}$.

Theorem 2. *For a hybrid automaton H, an STL formula φ, a time t, a time bound τ, and an arbitrary positive real value ϵ, the following properties are hold:*
(1) $\exists \sigma \in H.\ \sigma, t \models_\tau \neg(\varphi^{+\epsilon})$ *implies* $\exists \sigma \in H.\ \rho_\tau(\neg\varphi, \sigma, t) \geq -\epsilon$, *and*
(2) $\forall \sigma \in H.\ \sigma, t \not\models_\tau \neg(\varphi^{+\epsilon})$ *implies* $\forall \sigma \in H.\ \rho_\tau(\varphi, \sigma, t) \geq \epsilon$.

Proof. (1) Proof by contradiction. Suppose $\forall \sigma \in H$, $\rho_\tau(\neg\varphi, \sigma, t) < -\epsilon$. Then, $\forall \sigma \in H$, $\sigma, t \not\models_\tau (\neg\varphi)^{-\epsilon}$ by Lemma 1.(2). It is equivalent to $\forall \sigma \in H$, $\sigma, t \models_\tau \varphi^{+\epsilon}$, since $(\neg\varphi)^{-\epsilon} \equiv \neg(\varphi^{+\epsilon})$ by definition. It contradicts the assumption $\exists \sigma \in H.\ \sigma, t \models_\tau \neg(\varphi^{+\epsilon})$.
(2) Proof by contradiction. Suppose $\exists \sigma \in H$, $\rho_\tau(\varphi, \sigma, t) < \epsilon$.
Then, $\exists \sigma \in H$, $\sigma, t \not\models_\tau \varphi^{+\epsilon}$ by Lemma 2.(1). It contradicts the assumption $\forall \sigma \in H.\ \sigma, t \not\models_\tau \neg(\varphi^{+\epsilon})$, since it is equal to $\forall \sigma \in H.\ \sigma, t \models_\tau \varphi^{+\epsilon}$.

As a consequence, a counterexample for Boolean STL model checking of $(\neg\varphi)^{-\epsilon}$ is also a counterexample for robust STL model checking of φ with respect to ϵ. If there is no counterexample of $(\neg\varphi)^{-\epsilon}$, then it is guaranteed that the robustness degree of φ is always greater than or equal to ϵ. Therefore, for any robustness threshold $0 < \epsilon' < \epsilon$, φ is satisfied at t in H with respect to ϵ'. It is worth noting that φ may not be satisfied with respect to ϵ itself.

4.2 Boolean STL Bounded Model Checking

For Boolean STL model checking, there exist *refutationally complete* bounded model checking algorithms [7, 33]. There are two bound parameters for these algorithms: a *time bound* τ that restricts a time domain of a hybrid automaton, and a *discret bound* N that restricts the number of mode changes and the number of variable points at which the truth value of some STL subformula changes. There exists an SMT encoding $\Psi^{n,\tau}_{H,(\neg\varphi)}$, which is satisfiable if there exists a counterexample trajectory of H up to bounds N and τ:

Theorem 3. *[7, 33] There exists a trajectory $\sigma \in H$ with at most N variable points and mode changes such that $\sigma, t \not\models_\tau \varphi$ iff $\Psi^{N,\tau}_{H,(\neg\varphi)}$ is satisfiable.*

The satisfiability of $\Psi^{N,\tau}_{H,(\neg\varphi)}$ can be (approximately) determined using an SMT solver. When the flow conditions are linear or polynomial, the satisfiability can be precisely determined using Z3 [15] and Yices2 [20]. For hybrid automata with ODE dynamics, the satisfiability is undecidable in general, but a solver like dReal [26] can approximately determine its satisfiability.

Algorithm 1: Two-Step SMT Solving Algorithm

Input: Hybrid automaton H, STL formula φ, threshold ϵ, bounds τ and N

```
for k = 1 to N do
    Ψ̄ ← abstraction of the encoding Ψ^{k,τ}_{H,¬(φ^{+ε})} without flow and inv;
    while checkSat(Ψ̄) is Sat do
        π ← a minimal satisfying scenario;
        π̂ ← the refinement of π with flow and inv;
        if checkSat(π̂) is Sat then
            return counterexample(result.satAssignment);
        Ψ̄ ← Ψ̄ ∧ ¬π;
return True;
```

4.3 Two-Step Solving Algorithm

The computation cost of dReal for ODEs is highly expensive. To reduce the complexity, we propose a two-phase SMT solving algorithm, summariazed in Alg. 1, in a similar way to the lazy SMT solving approach [42]:

1. We abstract the flow and invariant conditions from the encoding $\Psi^{k,\tau}_{H,(\neg\varphi)^{-\epsilon}}$ using Boolean variables. We then enumerate a satisfiable assignment using an SMT solver to obtain a possible *scenario* π, given by a conjunction of literals, without taking into account the continuous dynamics.
2. For each π, we check the satisfiability using dReal considering the flow and invariant conditions. If any conjunction is satisfiable, we obtain a counterexample. If all conjunctions are unsatisfiable by dReal, it is guaranteed that there is no counterexample.

A naive implementation of the two-phase solving algorithm may generate many redundant scenarios for hybrid automata. Consider three jump conditions with guards $x > 60$, $y > 10$, and $z > 20$. There are 7 possible scenarios from these guards for a single jump. However, when one jump, say $x > 60$, is taken, the guards for the other jumps, i.e., $y > 10$ and $z > 20$, are not important. It suffices to consider only 3 scenarios, namely, $x > 60$, $y > 10$, and $z > 20$.

We implement a simple algorithm to minimize scenarios. A conjunction of literals π is a scenario if π implies the encoding Ψ. The scenario is minimized if Ψ is satisfied when π is false.

A scenario $\pi = l_1 \wedge \cdots \wedge l_m$ is minimal if Ψ is unsatisfied when $(\neg l_i \wedge \bigwedge_{j \neq i} l_j)$ is satisfied, where a literal in π is false. To minimize a scenario π, we apply a dual propagation approach [37]. Because π implies Ψ, the formula $\pi \wedge \neg\Psi$ is unsatisfiable. We evaluate the unsatisfiable core of $\pi \wedge \neg\Psi$ using Z3 to extract a minimal scenario from π.

4.4 Implementation

Figure 4 shows the overall architecture of STLmc. The given STL formula φ is first translated into the ϵ-weakening of the negated formula $(\neg\varphi)^{-\epsilon}$. The SMT

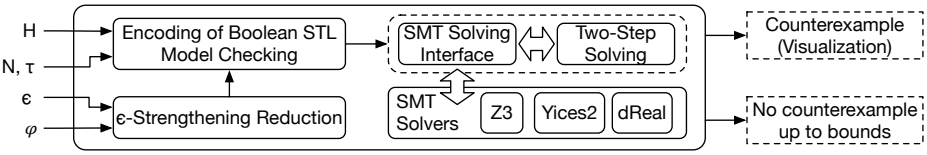

Fig. 4: The STLMC architecture

encoding $\Psi^{k,\tau}_{H,(\neg\varphi)^{-\epsilon}}$ for $1 \leq k \leq N$ is then built using the STL bounded model checking algorithm [7,33]. The satisfiability of $\Psi^{k,\tau}_{H,(\neg\varphi)^{-\epsilon}}$ can be checked directly using an SMT solver or using the two-phase optimization algorithm. Our tool is implemented in around 9,500 lines of Python code.

Simplifying Universal Quantification. The encoding $\Psi^{k,\tau}_{H,(\neg\varphi)^{-\epsilon}}$ includes universal quantification over time. Such $\exists\forall$-conditions are supported by several solvers but involve high computational costs. STLMC implements the following methods to simplify universal quantification. For polynomial hybrid automata, we use the quantifier-free encoding [13] to encode $\exists\forall$-conditions as quantifier-free formulas. For invariant STL properties of the form $p \rightarrow \Box_I q$, we reduce redundant universal quantifiers by reducing the model checking problem into the reachability [33].

Parallelizing Two-Phase Solving. We parallelize the two-phase solving algorithm for ODEs. As mentioned, the first phase generates abstract scenarios, and the second phase checks the feasibility of the scenarios. We simply run the feasibility checking of different scenarios in parallel. If any of such scenario is satisfied, then a counterexample is found and all other jobs can be terminated. If all scenarios, checking in parallel, are unsatisfiable, then there is no counterexample. As shown in Sec 6, it greatly improves the performance for the ODE cases in practice.

Supporting Various SMT Solvers. We implement a generic wrapper interface based on the SMT2LIB standard [9] to support various SMT solvers. Therefore, it is easy to extend our tool with a new SMT solvers, if it follows SMT2LIB. Given an input model, STLMC can detect the most suitable solvers. If the model has ODE dynamics and Z3 is selected, the tool raises an exception. Also, when the model has linear dynamics and the dReal is chosen, STLMC gives a warning to use Z3 or Yices instead of dReal.

5 Case Studies

5.1 Load Management of Batteries

There are two fully charged batteries and a control system balancing the load between these batteries to achieve longer lifetime (adapted from [23]). The total charge g_i and kinetic energy d_i of the battery $i = 1, 2$ changes according to the

controller's modes: *on* (On), *off* (Off), and *dead* (Dead). The dynamics for each battery is given as follows:

$$\dot{d}_i = \begin{cases} L/c - 0.5d_i & (\mathsf{On}) \\ -0.5d_i & (\mathsf{Off}) \\ 0 & (\mathsf{Dead}) \end{cases} \qquad \dot{g}_i = \begin{cases} -L & (\mathsf{On}) \\ 0 & (\mathsf{Off}) \\ 0 & (\mathsf{Dead}) \end{cases}$$

where L is load of the battery and $c \in [0, 1]$ is a threshold constant. If the charge g_i greater than $(1 - c)d_i$, the battery is dead. Otherwise, it can be either on or off. For simplification, we use the following dynamics:

$\dot{d}_1 = 1,$	$\dot{g}_1 = -0.3,$	$\dot{d}_2 = 1,$	$\dot{g}_2 = -0.3$	$(\mathsf{On_1On_2})$
$\dot{d}_1 = 0.8,$	$\dot{g}_1 = -0.5,$	$\dot{d}_2 = -0.166,$	$\dot{g}_2 = 0$	$(\mathsf{On_1Off_2})$
$\dot{d}_1 = -0.166,$	$\dot{g}_1 = 0,$	$\dot{d}_2 = 0.8,$	$\dot{g}_2 = -0.5$	$(\mathsf{Off_1On_2})$
$\dot{d}_1 = 0.7,$	$\dot{g}_1 = -7,$	$\dot{d}_2 = 0,$	$\dot{g}_2 = 0$	$(\mathsf{On_1Dead_2})$
$\dot{d}_1 = 0,$	$\dot{g}_1 = 0,$	$\dot{d}_2 = 0.7,$	$\dot{g}_2 = -7$	$(\mathsf{Dead_1On_2})$
$\dot{d}_1 = 0,$	$\dot{g}_1 = 0,$	$\dot{d}_2 = 0,$	$\dot{g}_2 = 0$	$(\mathsf{Dead_1Dead_2})$

Figure 5 shows a hybrid automaton of the battery system. The controller starts to use both batteries as an energy source, selecting at least one battery until both of them become dead. Table 2 shows three STL formulas for the battery system.

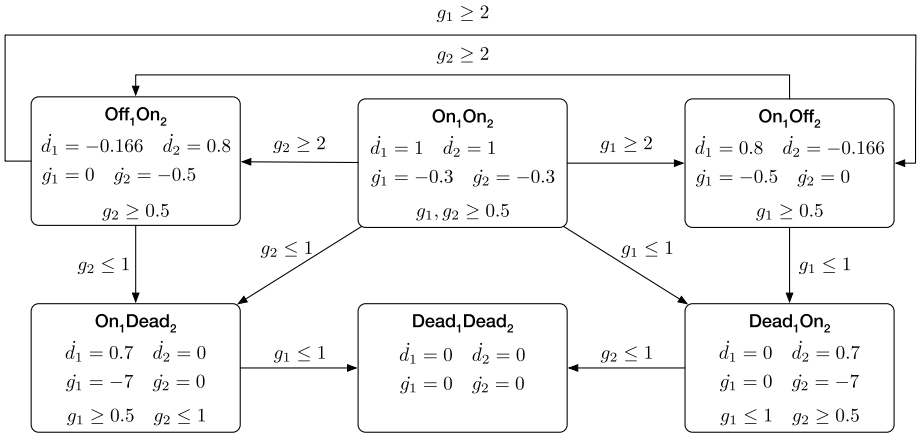

Fig. 5: A hybrid automaton of the battery system

Table 2: STL properties for the battery

Label	STL formula	Description
f1:	$\Diamond_{[4,10]}(d_2 \geq 4 \rightarrow \Box_{[4,10]}(b_2 = 2))$	For some time point in interval $[4, 10]$, if d_2 is greater than or equal to 4, then b_2 is always equal to 2 during time interval $[4, 10]$.
f2:	$(\Diamond_{[1,5]}\, g_2 \leq 5)\, \mathbf{R}_{[5,20]}(d_2 \geq 4.5)$	Within time interval $[5, 20]$, d_2 is greater than or equal to 4.5 until g_2 is less than or equal to 5 for some time point in interval $[1, 5]$.
f3:	$\Box_{[4,14]}(g_2 > 4 \rightarrow \Diamond_{[0,10]}(d_2 > 1))$	g_1 is greater than 1 for 4 time units, until d_1 is greater than or equal to 2.

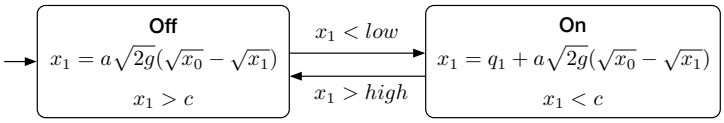

Fig. 6: A hybrid automaton of one water tank controller H_1

5.2 Two Networked Water Tank Systems

Two water tanks are connected by pipes, where water flows from one tank to another (adapted from [38]). The water level x_i of tank $i = 0, 1$ is controlled by its pump. There are two pump modes: On and Off. Each pump changes its mode according to the water level of its tank and the other water tank. The continuous dynamics of x_i is defined as follows:

$$\dot{x}_i = \begin{cases} q_i + a\sqrt{2g}(\sqrt{x_{i-1}} - \sqrt{x_i}) & (\mathsf{On}) \\ a\sqrt{2g}(\sqrt{x_{i-1}} - \sqrt{x_i}) & (\mathsf{Off}) \end{cases}$$

where a and q_i depend on the width of the pipe and the pump's power, and g is the standard gravity constant. Figure 6 shows a hybrid automaton H_1 of one water tank (the other component is similar). For simplification, we consider the following dynamics,

$$\dot{x}_0 = \begin{cases} 0.9 & (\mathsf{On}) \\ -0.8 & (\mathsf{Off}) \end{cases} \qquad \dot{x}_1 = \begin{cases} 1 & (\mathsf{On}) \\ -0.6 & (\mathsf{Off}) \end{cases}$$

Table 3 shows three STL formulas for the water tank systems.

5.3 Autonomous Driving of Two Cars

We consider two autonomous cars: one drives according to its own scenario, and the other follows the leader (adapted from [3]). The *relative* distance of the

Label	STL formula	Description
f1:	$\Box_{[1,3]}((x_0 \leq 7)\,\mathbf{R}_{[1,10]}(x_1 \leq 3))$	For time interval $[1,3]$, x_1 is less than or equal to 3, until x_0 is less than or equal to 7 for time interval $[1,10]$.
f2:	$(\Diamond_{[1,10]}(x_1 < 5.5))\,\mathbf{U}_{[2,5]}(x_0 \geq 5)$	Within time interval $[1,10]$, x_1 is less than 5.5, until x_0 is greater than or equal to 5 in time interval $[2,5]$.
f3:	$(\Diamond_{[4,10]}(x_0 \geq 4 \rightarrow \Box_{[2,5]} x_1 \leq 2)$	For some time point in interval $[4,10]$, if x_0 is greater than or equal to 4 then x_1 is always less than or equal to 2 during time interval $[2,5]$.

Table 3: STL properties for the water tank systems

following car (r_x, r_y) depends on the modes of the car (CxCy, CxFy, FxCy, and FxFy) and its velocity (v_x, v_y). When the relative distance of the car is too far (or close), the car accelerates (or decelerates). For each mode, we consider the following dynamics:

$$\begin{aligned}
&\dot{r}_x = v_x, \quad &&\dot{r}_y = v_y, \quad &&\dot{v}_x = 1.2, \quad &&\dot{v}_y = 1.4 \quad &&\text{(CxCy)}\\
&\dot{r}_x = v_x, \quad &&\dot{r}_y = v_y, \quad &&\dot{v}_x = 1.2, \quad &&\dot{v}_y = -1.4 \quad &&\text{(CxFy)}\\
&\dot{r}_x = v_x, \quad &&\dot{r}_y = v_y, \quad &&\dot{v}_x = -1.2, \quad &&\dot{v}_y = 1.4 \quad &&\text{(FxCy)}\\
&\dot{r}_x = v_x, \quad &&\dot{r}_y = v_y, \quad &&\dot{v}_x = -1.2, \quad &&\dot{v}_y = -1.4 \quad &&\text{(FxFy)}
\end{aligned}$$

Figure 7 shows a hybrid automaton of the autonomous car. Initially, the relative distance is close. The following car tries to maintain the distance within certain range according to the control logic. Table 4 shows three STL formulas for the autonomous cars.

5.4 A Railroad Gate Controller

There are a gate and a gate controller with a crossing bar on a circular railroad track and a train moves on the track. The gate controller opens (or closes) the crossing bar when the train approaches to (or leaves from) the gate (adapted from [28]). Figure 8 shows a hybrid automaton for this system. The position of the crossing bar p_b from the ground changes according to the velocity of bar v_b, depending on the modes (Far, Approach, Close, and Past) of the gate controller and the distance x, between the gate and the train. For each mode, we consider the following dynamics.

$$\begin{aligned}
&\dot{x} = -30, \quad &&\dot{p}_b = v_b, \quad &&\dot{v}_b = 0 \quad &&\text{(Far)}\\
&\dot{x} = -5, \quad &&\dot{p}_b = v_b, \quad &&\dot{v}_b = 0.3 \quad &&\text{(Approach)}\\
&\dot{x} = -5, \quad &&\dot{p}_b = v_b, \quad &&\dot{v}_b = 0.5 \quad &&\text{(Close)}\\
&\dot{x} = -5, \quad &&\dot{p}_b = v_b, \quad &&\dot{v}_b = -1 \quad &&\text{(Past)}
\end{aligned}$$

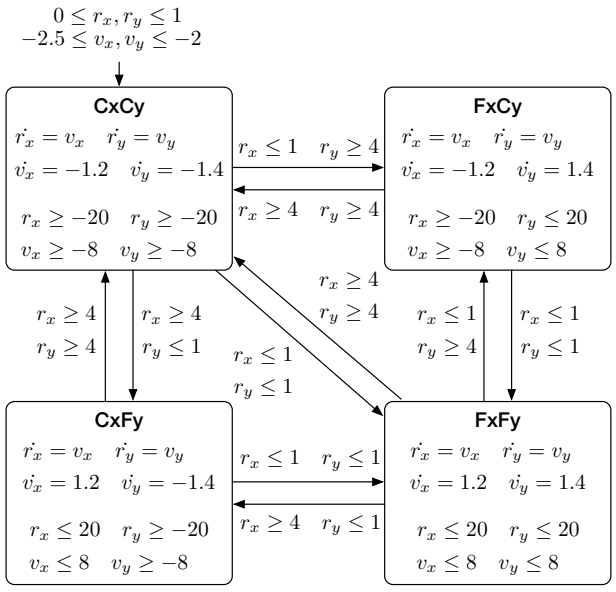

Fig. 7: A hybrid automaton of the autonomous car (F: Far, C: Close)

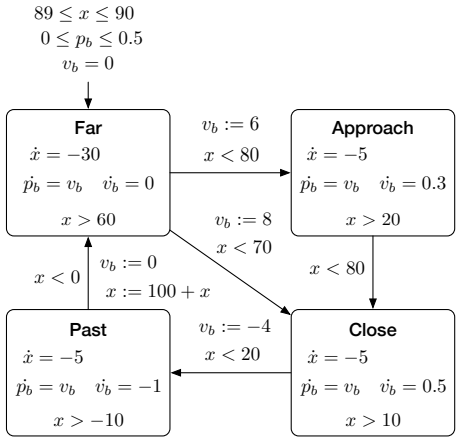

Fig. 8: A hybrid automaton of the railroad gate controller

Table 4: STL properties for the autonomous car

Label	STL formula	Description
f1:	$\Box_{[0,4]}(v_x < -2 \rightarrow \Diamond_{[2,5]}\, r_x \leq -2)$	For the first 4 time units, if v_x is less than -2, then r_x is greater than or equal to -2 for some time in interval $[2, 5]$.
f2:	$(\Diamond_{[0,4]}\, r_y > 3)\, \mathbf{U}_{[0,5]}(v_y > 1.5)$	For some time point in interval $[0, 4]$, r_y is greater than 3 until v_y is greater than 1.5 for time interval $[0, 5]$.
f3:	$\Diamond_{[0,3]}(r_x \leq 5\, \mathbf{U}_{[0,5]}\, ix = false)$	r_x is less than or equal to 5 until ix become $false$ for time interval $[0, 5]$, for some time point in interval $[0, 3]$.

Table 5 shows three STL formulas for the railroad gate controller.

Label	STL formula	Description
f1:	$\Diamond_{[0,5]}((p_b \geq 40)\, \mathbf{U}_{[1,8]}(x < 40))$	For some time point in interval $[0, 5]$, p_b is greater than or equal to 40, until x is less than 40 during time interval $[1, 8]$.
f2:	$\Diamond_{[0,4]}(x < 50 \rightarrow \Box_{[2,10]}(p_b > 40))$	Within 4 time units, if x is less than 50, then p_b is greater than 40 for time interval $[2, 10]$.
f3:	$\Box_{[0,5)}((x < 50)\, \mathbf{U}_{[2,10]}(p_b > 5))$	During time interval $[0, 5)$, x is less than 50 until p_b is greater than 5 within time interval $[2, 10]$.

Table 5: STL properties for the railroad gate controller

5.5 A Filtered Oscillator

We consider a filtered oscillator model adapted from [24]. The filtered oscillator consists of a 2-dimensional switched oscillator of signals x and y with a k-series of first-order filters. The filters smoothens an input signal x, producing an output signal z. When the signal x goes through each filter, the amplitude of the signal diminishes as it passing by. Each filter $i \in \{0, 1, 2, 3\}$ takes an input signal x_i and outputs a diminished signal x_{i+1}, where $x_0 = x$ and $x_4 = z$. Figure 9 shows a hybrid automaton of the filtered oscillator. Initially, the x and y are in $[0.2, 0.3]$ and $[-0.1, 0.1]$, respectively and all filters output zero signals. The

jumps between modes define the behavior of the filtered oscillator to maintain a stable oscillation.

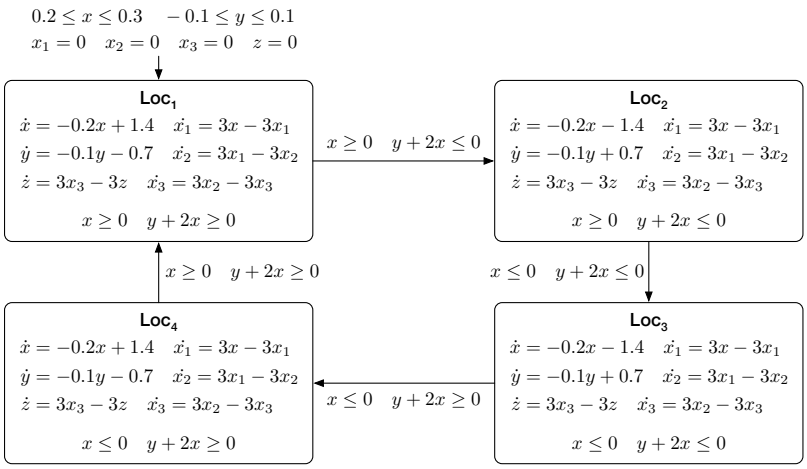

Fig. 9: A hybrid automaton of the filtered oscillator

Table 6 shows three STL formulas for the filtered oscillator.

Table 6: STL properties for the oscillator

Label	STL formula	Description
f1:	$\Diamond_{[0,3]}((x_3 \ge 1)\, \mathbf{R}_{[0,\infty)}(y \le 10))$	For some time point in $[0,3]$, y is less than or equal to 10, until x_3 is greater than or equal to 1 within time interval $[0,\infty)$.
f2:	$\Diamond_{[2,5]}(\Box_{[0,3]}(x_2 < 4))$	For some time point in $[2,5]$, x_2 is always less than 4 during 3 time units.
f3:	$(\Box_{[1,3]}(x \le 2))\, \mathbf{R}_{[2,5]}(x_3 > 2)$	For time interval $[2,5]$, x_3 is greater than 2, until x is less than or equal to 2 within time interval $[1,3]$.

6 Experimental Evaluation

We evaluate the effectiveness of the STLMC model checker using a number of hybrid system benchmarks and nontrivial STL properties. We use the following

Table 7: Robust Bounded Model Checking of STL (Time in seconds)

Dyn.	Model	STL formula	ϵ	$\lvert\Psi\rvert$	Time	Result	k	Alg.	$\#\pi$
Linear (N = 20)	Bat	$\Diamond_{[4,10]}(p_1 \rightarrow \Box_{[4,10]}\, p_2)$	0.1	12.9	137	$\top$	-	1-step	-
		$(\Diamond_{[1,5]}\, p_1)\, \mathbf{R}_{[5,20]}\, p_2$	3.5	2.76	5.71	$\bot$	5	1-step	-
		$\Box_{[4,14]}(p_1 \rightarrow \Diamond_{[0,10]}\, p_2)$	0.1	3.8	22.1	$\bot$	8	1-step	-
	Water	$\Box_{[1,3]}(p_1\, \mathbf{R}_{[1,10]}\, p_2)$	2.5	18.8	26.2	$\top$	-	1-step	-
		$(\Diamond_{[1,10]}\, p_1)\, \mathbf{U}_{[2,5]}\, p_2$	0.1	1.9	4.22	$\bot$	4	1-step	-
		$\Diamond_{[4,10]}(p_1 \rightarrow \Box_{[2,5]}\, p_2)$	0.01	11.2	20.2	$\top$	-	1-step	-
Poly (N = 10)	Car	$\Box_{[0,4]}(p_1 \rightarrow \Diamond_{[2,5]}\, p_2)$	0.5	2.2	7.24	$\bot$	5	1-step	-
		$(\Diamond_{[0,4]}\, p_1)\, \mathbf{U}_{[0,5]}\, p_2$	2.0	1.7	6.27	$\bot$	3	1-step	-
		$\Diamond_{[0,3]}(p_1\, \mathbf{U}_{[0,5]}\, p_2)$	0.1	7.3	9.72	$\top$	-	1-step	-
	Rail	$\Diamond_{[0,5]}(p_1\, \mathbf{U}_{[1,8]}\, p_2)$	1.0	2.3	3.43	$\bot$	5	1-step	-
		$\Diamond_{[0,4]}(p_1 \rightarrow \Box_{[2,10]}\, p_2)$	5.0	3.8	0.86	$\top$	-	1-step	-
		$(\Box_{[0,5)}\, p_1)\, \mathbf{U}_{[2,10]}\, p_2$	4.0	1.9	2.83	$\bot$	4	1-step	-
ODE (N = 5)	Therm	$\Diamond_{[0,3]}(p_1\, \mathbf{U}_{[0,\infty)}\, p_2)$	1.0	1.2	817	$\top$	-	2-step	3,646
		$\Box_{[2,4]}(p_1 \rightarrow \Diamond_{[3,10]}\, p_2)$	2.0	0.7	7.46	$\bot$	2	2-step	47
		$\Box_{[0,10]}(p_1\, \mathbf{R}_{[0,\infty)}\, p_2)$	2.0	1.2	59.3	$\bot$	4	2-step	212
	Oscil	$\Diamond_{[0,3]}(p_1\, \mathbf{R}_{[0,\infty)}\, p_2)$	0.1	1.5	110	$\top$	-	2-step	289
		$\Diamond_{[2,5]}(\Box_{[0,3]}\, p_1)$	1.0	1.2	224	$\bot$	3	2-step	259
		$(\Box_{[1,3]}\, p_1)\, \mathbf{R}_{[2,5]}\, p_2$	0.1	1.2	266	$\bot$	3	2-step	266

models, adapted from existing benchmarks [3, 6, 23, 24, 28, 38]: load management for two batteries (Bat), two networked water tank systems (Water), autonomous driving of two cars (Car), a railroad gate (Rail), two networked thermostats (Therm), and a filtered oscillator (Oscil). We use a modified model with either linear, polynomial, or ODE dynamics to analyze the effect of different continuous dynamics. For each model, we use three STL formulas with nested temporal operators. The models and the experimental results are available in [46].

We measure the SMT encoding size and execution time for robust STL model checking, up to discrete bound $N = 20$ for linear models, $N = 10$ for polynomial models, and $N = 5$ for ODEs models, with a timeout of 30 minutes. We use different time bounds τ and robustness thresholds ϵ for different models, since τ and ϵ depend on each model. As an underlying SMT solver, we use Yices for linear and polynomial models, and dReal for ODE models with a precision $\delta = 0.001$. We run both direct SMT solving (1-step) and two-step SMT solving (2-step). We use 25 cores for parallelizing the two-phase solving algorithm. We have run all experiments on Intel Xeon 2.8GHz with 256 GB memory.

The experimental results are summarized in Table 7, where $\lvert\Psi\rvert$ denotes the size of the SMT encoding Ψ (in thousands) as the number of connectives in Ψ. For the model checking results, $\top$ indicates that the tool found no counterexample up to bound N, and $\bot$ indicates that the tool found a counterexample at bound $k \leq N$. For the algorithms (Alg.), we write one of the results with a better performance. For the 2-step case, we also write the number of minimal scenarios

generated ($\#\pi$). Actually, two-step SMT solving timed out for all linear and polynomial models, and direct SMT solving timed out for all ODE models.

As shown in Table 7, our tool can perform robust model checking of nontrivial STL formulas for hybrid systems with different continuous dynamics. The cases of ODE models generally take longer than the cases of linear and polynomial models, because of the high computational costs for ODE solving. Nevertheless, our parallelized two-step SMT solving method works well and all model checking analyses are finished before the timeout. In contrast, for linear and polynomial models with a larger discrete bound $N \geq 10$, direct SMT solving is usually effective but the two-step SMT solving method is not. There are too many scenarios, and the scenario generation does not terminate within 30 minutes. Therefore, the two algorithms implemented in our tool are complementary.

7 Related Work

There exist many tools for falsifying STL properties of hybrid systems, including Breach [17], S-talrio [4], and TLTk [14]. STL falsification techniques are based on STL monitoring [16, 18, 29, 36], and often use stochastic optimization techniques, such as Ant-Colony Optimization [4], Monte-Carlo tree search [47], deep reinforcement learning [2], and so on. These techniques are often quite useful for finding counterexamples in practice, but, as mentioned, cannot be used to verify STL properties of hybrid systems.

There exist many tools for analyzing reachability properties of hybrid systems based on reachable-set computation, including C2E2 [19], Flow* [11], Hylaa [8], and SpaceEx [24]. They can be used to guarantee the correctness of invariant properties of the form $p \rightarrow \Box_I q$, but cannot verify general STL properties. In contrast, STLMC uses a refutation-complete bounded STL model checking algorithm to verify general STL properties, including complex ones.

Our tool is also related to SMT-based tools for analyzing hybrid systems, including dReach [32], HyComp [12], and HybridSAL [43]. These techniques also focus on analyzing invariant properties of hybrid systems, but some SMT-based tools, such as HyComp, can verify LTL properties of hybrid systems. Unlike STLMC, they cannot deal with general STL properties of hybrid systems.

8 Concluding Remarks

We have presented the STLMC tool for robust bounded model checking of STL properties for hybrid systems. STLMC can verify that, up to given bounds, the robustness degree of an STL formula φ is always greater than a given robustness threshold for all possible behaviors of a hybrid system. STLMC also provides a convenient user interface with an intuitive counterexample visualization.

Our tool leverages the reduction from robust model checking to Boolean model checking, and utilizes the refutation-complete SMT-based Boolean STL model checking algorithm to guarantee correctness up to given bounds and find subtle counterexamples. STLMC can deal with hybrid systems with (nonlinear)

ODEs using dReal. We have shown using various hybrid system benchmarks that STLMC can effectively analyze nontrivial STL properties.

Future work includes extending our tool with other hybrid system analysis methods, such as reachable-set computation, besides SMT-based approaches.

References

1. Abbas, H., Fainekos, G., Sankaranarayanan, S., Ivančić, F., Gupta, A.: Probabilistic temporal logic falsification of cyber-physical systems. ACM Transactions on Embedded Computing Systems (TECS) **12**(2s), 1–30 (2013)
2. Akazaki, T., Liu, S., Yamagata, Y., Duan, Y., Hao, J.: Falsification of cyber-physical systems using deep reinforcement learning. In: International Symposium on Formal Methods. pp. 456–465. Springer (2018)
3. Alur, R.: Principles of cyber-physical systems. The MIT Press (2015)
4. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-taliro: A tool for temporal logic falsification for hybrid systems. In: Proc. TACAS. LNCS, vol. 6605, pp. 254–257. Springer (2011)
5. Bae, K., Gao, S.: Modular smt-based analysis of nonlinear hybrid systems. In: Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design. pp. 180–187. FMCAD '17, FMCAD, Austin, TX (2017), `http://dl.acm.org/citation.cfm?id=3168451.3168490`
6. Bae, K., Gao, S.: Modular SMT-based analysis of nonlinear hybrid systems. In: Proc. FMCAD. pp. 180–187. IEEE (2017)
7. Bae, K., Lee, J.: Bounded model checking of signal temporal logic properties using syntactic separation. Proc. ACM Program. Lang. **3, POPL**(51), 1–30 (2019)
8. Bak, S., Duggirala, P.S.: Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In: Proc. HSCC. pp. 173–178. ACM (2017)
9. Barrett, C., Stump, A., Tinelli, C., et al.: The SMT-LIB standard: Version 2.0. In: Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, England). vol. 13, p. 14 (2010)
10. Chen, G., Liu, M., Kong, Z.: Temporal-logic-based semantic fault diagnosis with time-series data from industrial internet of things. IEEE Transactions on Industrial Electronics **68**(5), 4393–4403 (2020)
11. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Proc. CAV. LNCS, vol. 8044, pp. 258–263. Springer (2013)
12. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: HyComp: an SMT-based model checker for hybrid systems. In: Proc. TACAS. LNCS, vol. 9035. Springer (2015)
13. Cimatti, A., Mover, S., Tonetta, S.: A quantifier-free smt encoding of non-linear hybrid automata. In: Formal Methods in Computer-Aided Design (FMCAD), 2012. pp. 187–195. IEEE (2012)
14. Cralley, J., Spantidi, O., Hoxha, B., Fainekos, G.: Tltk: A toolbox for parallel robustness computation of temporal logic specifications. In: International Conference on Runtime Verification. pp. 404–416. Springer (2020)
15. De Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Proc. TACAS. LNCS, vol. 4963, pp. 337–340. Springer (2008)
16. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. Form. Methods Syst. Des. **51**(1) (2017)
17. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Proceedings of the 22nd International Conference on Computer Aided Verification. pp. 167–170. CAV'10, Springer (2010)

18. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Proc. CAV. LNCS, vol. 8044. Springer (2013)
19. Duggirala, P.S., Mitra, S., Viswanathan, M., Potok, M.: C2e2: A verification tool for stateflow models. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 68–82. Springer (2015)
20. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) Proc. CAV. LNCS, vol. 8559, pp. 737–744. Springer (2014)
21. Eddeland, J.L., Donzé, A., Miremadi, S., Åkesson, K.: Industrial temporal logic specifications for falsification of cyber-physical systems. ARCH (2020)
22. Farahani, S.S., Soudjani, S.E.Z., Majumdar, R., Ocampo-Martinez, C.: Robust model predictive control with signal temporal logic constraints for barcelona wastewater system. IFAC-PapersOnLine **50**(1), 6594–6600 (2017)
23. Fox, M., Long, D., Magazzeni, D.: Plan-based policies for efficient multiple battery load management. JAIR **44**, 335–382 (2012)
24. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Proc. CAV. LNCS, vol. 6806, pp. 379–395. Springer (2011)
25. Gao, S., Avigad, J., Clarke, E.M.: Delta-decidability over the reals. In: 2012 27th Annual IEEE Symposium on Logic in Computer Science. pp. 305–314. IEEE (2012)
26. Gao, S., Kong, S., Clarke, E.M.: dReal: An SMT solver for nonlinear theories over the reals. In: Proc. CADE. LNCS, vol. 7898, pp. 208–214. Springer (2013)
27. Goldman, R.P., Bryce, D., Pelican, M.J., Musliner, D.J., Bae, K.: A hybrid architecture for correct-by-construction hybrid planning and control. In: Proc. NFM. LNCS, vol. 9690. Springer (2016)
28. Henzinger, T.: The theory of hybrid automata. In: Verification of Digital and Hybrid Systems, NATO ASI Series, vol. 170, pp. 265–292. Springer (2000)
29. Jakšić, S., Bartocci, E., Grosu, R., Ničković, D.: Quantitative monitoring of STL with edit distance. Form. Methods Syst. Des. **53**(1), 83–112 (2018)
30. Jin, X., Deshmukh, J.V., Kapinski, J., Ueda, K., Butts, K.: Powertrain control verification benchmark. In: Proc. HSCC. ACM (2014)
31. Kapinski, J., Deshmukh, J.V., Jin, X., Ito, H., Butts, K.: Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques. IEEE Control Systems Magazine **36**(6), 45–64 (2016)
32. Kong, S., Gao, S., Chen, W., Clarke, E.M.: dReach: δ-reachability analysis for hybrid systems. In: Proc. TACAS. LNCS, vol. 7898, pp. 200–205. Springer (2015)
33. Lee, J., Yu, G., Bae, K.: Efficient smt-based model checking for signal temporal logic. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 343–354 (2021). https://doi.org/10.1109/ASE51524.2021.9678719
34. Ma, M., Bartocci, E., Lifland, E., Stankovic, J., Feng, L.: Sastl: spatial aggregation signal temporal logic for runtime monitoring in smart cities. In: 2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS). pp. 51–62. IEEE (2020)
35. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Proc. FORMATS. LNCS, vol. 3253, pp. 152–166. Springer (2004)
36. Ničković, D., Lebeltel, O., Maler, O., Ferrère, T., Ulus, D.: Amt 2.0: qualitative and quantitative trace analysis with extended signal temporal logic. In: Proc. TACAS. vol. 10806, pp. 303–319. Springer (2018)

37. Niemetz, A., Preiner, M., Biere, A.: Turbo-charging lemmas on demand with don't care reasoning. In: 2014 Formal Methods in Computer-Aided Design (FMCAD). pp. 179–186. IEEE (2014)
38. Raisch, J., Klein, E., Meder, C., Itigin, A., O'Young, S.: Approximating automata and discrete control for continuous systems — two examples from process control. In: Hybrid systems V. LNCS, vol. 1567, pp. 279–303. Springer (1999)
39. Roehm, H., Oehlerking, J., Heinz, T., Althoff, M.: STL model checking of continuous and hybrid systems. In: Proc. ATVA. LNCS, vol. 9938. Springer (2016)
40. Roohi, N., Kaur, R., Weimer, J., Sokolsky, O., Lee, I.: Parameter invariant monitoring for signal temporal logic. In: Proc. HSCC. pp. 187–196. ACM (2018)
41. Sankaranarayanan, S., Fainekos, G.: Falsification of temporal properties of hybrid systems using the cross-entropy method. In: Proceedings of ACM international conference on Hybrid Systems: Computation and Control. pp. 125–134 (2012)
42. Sebastiani, R.: Lazy satisfiability modulo theories. Journal on Satisfiability, Boolean Modeling and Computation **3**(3-4), 141–224 (2007)
43. Tiwari, A.: HybridSAL relational abstracter. In: Proc. CAV. Lecture Notes in Computer Science, vol. 7358, pp. 725–731. Springer (2012)
44. Wu, T., Olama, M.M., Djouadi, S.M., Dong, J., Xue, Y., Kuruganti, T.: Signal temporal logic control for residential hvac systems to accommodate high solar pv penetration. In: 2020 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT). pp. 1–5. IEEE (2020)
45. Xu, Z., Belta, C., Julius, A.: Temporal logic inference with prior information: An application to robot arm movements. IFAC-PapersOnLine **48**(27), 141–146 (2015)
46. Yu, G., Lee, J., Bae, K.: Supplementary material: technical report, benchmark models, and experiments (2022), `https://stlmc.github.io/cav2022`
47. Zhang, Z., Ernst, G., Sedwards, S., Arcaini, P., Hasuo, I.: Two-layered falsification of hybrid systems guided by Monte-Carlo tree search. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **37**(11) (2018)
48. Zhang, Z., Lyu, D., Arcaini, P., Ma, L., Hasuo, I., Zhao, J.: Effective hybrid system falsification using monte carlo tree search guided by qb-robustness. In: Proc. CAV. LNCS, vol. 12759, pp. 595–618. Springer (2021)

A STLmc Input Models

A.1 Load Management for Two Batteries

The battery system can be specified as an input model using the STLmc language (see Fig. 10).

```
real b1;    real b2;
[0, 10] g1; [0, 10] g2;
[0, 10] d2; [0, 10] d1;
# 1: ON, 2: OFF, 3: DEAD
{ mode: b1 = 1; b2 = 1;
  inv:  g1 >= 0.5; g2 >= 0.5;
  flow: d/dt[d1] = 1;
        d/dt[d2] = 1;
        d/dt[g1] = -0.3;
        d/dt[g2] = -0.3;
  jump: g1 >= 2 => (and (b1' = 1) (b2' = 2)
                        (d1' = d1) (d2' = d2)
                        (g1' = g1) (g2' = g2));
        g1 <= 1 => (and (b1' = 3) (b2' = 1)
                        (d1' = d1) (d2' = d2)
                        (g1' = g1) (g2' = g2));
        g2 >= 2 => (and (b1' = 2) (b2' = 1)
                        (d1' = d1) (d2' = d2)
                        (g1' = g1) (g2' = g2));
        g2 <= 1 => (and (b1' = 1) (b2' = 3)
                        (d1' = d1) (d2' = d2)
                        (g1' = g1) (g2' = g2));
}
{ mode: b1 = 1; b2 = 2;
  inv:  g1 >= 0.5;
  flow: d/dt[d1] = 0.8;
        d/dt[d2] = -0.166;
        d/dt[g1] = -0.5;
        d/dt[g2] = 0;
  jump: g2 >= 2 => (and (b1' = 2) (b2' = 1)
                        (d1' = d1) (d2' = d2)
                        (g1' = g1) (g2' = g2));
        g1 <= 1 => (and (b1' = 3) (b2' = 1)
                        (d1' = d1) (d2' = d2)
                        (g1' = g1) (g2' = g2));
}
{ mode: b1 = 2; b2 = 1;
  inv:  g2 >= 0.5;
  flow: d/dt[d1] = -0.166;
        d/dt[d2] = 0.8;
        d/dt[g1] = 0;
        d/dt[g2] = -0.5;
  jump: g1 >= 2 => (and (b1' = 1) (b2' = 2)
                        (d1' = d1) (d2' = d2)
                        (g1' = g1) (g2' = g2));
        g2 <= 1 => (and (b1' = 1) (b2' = 3)
                        (d1' = d1) (d2' = d2)
                        (g1' = g1) (g2' = g2));
}
{ mode: b1 = 3; b2 = 1;
  inv:  g1 <= 1; g2 >= 0.5;
  flow: d/dt[d1] = 0;
        d/dt[d2] = 0.7;
        d/dt[g1] = 0;
        d/dt[g2] = -7;
  jump: g2 <= 1 => (and (b1' = 3) (b2' = 3)
                        (d1' = d1) (d2' = d2)
                        (g1' = g1) (g2' = g2));
}
{ mode: b1 = 1; b2 = 3;
  inv:  g1 >= 0.5; g2 <= 1;
  flow: d/dt[d1] = 0.7;
        d/dt[d2] = 0;
        d/dt[g1] = -7;
        d/dt[g2] = 0;
  jump: g1 <= 1 => (and (b1' = 3) (b2' = 3)
                        (d1' = d1) (d2' = d2)
                        (g1' = g1) (g2' = g2));
}
{ mode: b1 = 3; b2 = 3;
  inv:
  flow: d/dt[d1] = 0;
        d/dt[d2] = 0;
        d/dt[g1] = 0;
        d/dt[g2] = 0;
  jump:
}

init: b1 = 1; b2 = 1;
      0 <= d1; d1 <= 0.1; 8.4 <= g1; g1 <= 8.5;
      0 <= d2; d2 <= 0.1; 7.4 <= g2; g2 <= 7.5;

proposition:

# bound: 20, timebound: 30, solver: yices
goal:
  [f1]: <>[4, 10] (d2 >= 4 -> [][4, 10] b2 = 2);
  [f2]: (<>[1, 5] g2 <= 5) R[5, 20] (d2 >= 4.5);
  [f3]: [][4, 14] (g2 > 4 -> <>[0, 10] d2 > 1);
```

Fig. 10: An input model of the load management of two batteries

A.2 Two Networked Watertank Systems

Figure 11 is a model file of the water tank systems in STLmc.

```
int on0;    int on1;
[0, 10] x0; [0, 10] x1;
{ mode: on0 = 0;      on1 = 0;
  inv:  x0 >= 1; x1 >= 1;
  flow: d/dt[x0] = -0.8;
        d/dt[x1] = -0.6;
  jump: x0 <= 2 => (and (on0' = 1) (on1' = on1)
                        (x0' = x0) (x1' = x1));
        x1 <= 3 => (and (on0' = on0) (on1' = 1)
                        (x0' = x0) (x1' = x1));
        x0 <= 2 => (and (on0' = 1) (on1' = 1)
                        (x0' = x0) (x1' = x1));
        x1 <= 3 => (and (on0' = 1) (on1' = 1)
                        (x0' = x0) (x1' = x1));
}
{ mode: on0 = 0;      on1 = 1;
  inv:  x0 >= 1; x1 <= 9;
  flow: d/dt[x0] = -0.8;
        d/dt[x1] = 1;
  jump: x1 >= 6 => (and (on0' = on0) (on1' = 0)
                        (x0' = x0) (x1' = x1));
        x0 <= 2 => (and (on0' = 1) (on1' = on1)
                        (x0' = x0) (x1' = x1));
        x1 >= 6 => (and (on0' = 1) (on1' = 0)
                        (x0' = x0) (x1' = x1));
        x0 <= 2 => (and (on0' = 1) (on1' = 0)
                        (x0' = x0) (x1' = x1));
}
{ mode: on0 = 1;      on1 = 0;
  inv:  x0 <= 8; x1 >= 1;
  flow: d/dt[x0] = 0.9;
        d/dt[x1] = -0.6;
  jump: x0 >= 5 => (and (on0' = 0) (on1' = on1)
                        (x0' = x0) (x1' = x1));
        x1 <= 3 => (and (on0' = on1) (on1' = 1)
                        (x0' = x0) (x1' = x1));
        x0 >= 5 => (and (on0' = 0) (on1' = 1)
                        (x0' = x0) (x1' = x1));
        x1 <= 3 => (and (on0' = 0) (on1' = 1)
                        (x0' = x0) (x1' = x1));
}
{ mode: on0 = 1;     on1 = 1;
  inv:  x0 <= 9; x1 <= 9;
  flow: d/dt[x0] = 0.9;
        d/dt[x1] = 1;
  jump: x0 >= 5 => (and (on0' = 0) (on1' = on1)
                        (x0' = x0) (x1' = x1));
        x1 >= 6 => (and (on0' = on0) (on1' = 0)
                        (x0' = x0) (x1' = x1));
        x0 >= 5 => (and (on0' = 0) (on1' = 0)
                        (x0' = x0) (x1' = x1));
        x1 >= 6 => (and (on0' = 0) (on1' = 0)
                        (x0' = x0) (x1' = x1));
}
init: on0 = 0; 4.4 <= x0; x0 <= 4.5;
      on1 = 0; 5.9 <= x1; x1 <= 6;

proposition:

# timebound : 20
goal:
  [f1]: [][1, 3]((x0 <= 7) R[1, 10] (x1 <= 3));
  [f2]: (<>[1, 10) x1 < 5.5) U[2, 5] (x0 >= 5);
  [f3]: <>[4, 10] (x0 >= 4 -> [][2, 5] x1 <= 2);
```

Fig. 11: An input model of the water tank systems

A.3 Autonomous Driving of Two Cars

Figure 12 is a model file of the autonomous car in STLmc.

```
bool ix;        bool iy;
[-40, 40] rx;   [-40, 40] ry;
[-30, 30] vx;   [-30, 30] vy;
#  acc, acc
{ mode: ix = true; iy = true;
  inv:  rx < 20; ry < 20;
        vx < 8; vy < 8;
  flow: d/dt[rx] = vx;
        d/dt[ry] = vy;
        d/dt[vx] = 1.2;
        d/dt[vy] = 1.4;
  jump: (and (rx >= 3) (ry < 3)) =>
        (and (ix' = false) (iy' = true)
                   (rx' = rx) (ry' = ry)
                   (vx' = 0) (vy' = 0));
        (and (rx < 3) (ry >= 3)) =>
        (and (ix' = true) (iy' = false)
                   (rx' = rx) (ry' = ry)
                   (vx' = 0) (vy' = 0));
}
#  car1 : acc, dec
{ mode: ix = true; iy = false;
  inv:  rx < 20;  ry > -20;
        vx < 8;  vy > -8;
  flow: d/dt[rx] = vx;
        d/dt[ry] = vy;
        d/dt[vx] = 1.2;
        d/dt[vy] = -1.4;
  jump: (and (rx >= 3) (ry >= 3)) =>
        (and (ix' = false) (iy' = false)
             (rx' = rx) (ry' = ry)
             (vx' = 0) (vy' = 0));
        (and (rx >= 3) (ry < 3)) =>
        (and (ix' = false) (iy' = true)
                   (rx' = rx) (ry' = ry)
                   (vx' = 0) (vy' = 0));
}
#  car1 : dec, acc
{ mode: ix = false; iy = true;
  inv:  rx > -20; ry < 20;
        vx > -8; vy < 8;
  flow: d/dt[rx] = vx;
        d/dt[ry] = vy;
        d/dt[vx] = -1.2;
        d/dt[vy] = 1.4;
  jump: (and (rx < 3) (ry < 3)) =>
        (and (ix' = true) (iy' = true)
             (rx' = rx) (ry' = ry)
             (vx' = 0) (vy' = 0));
        (and (rx >= 3) (ry >= 3)) =>
        (and (ix' = false) (iy' = false)
             (rx' = rx) (ry' = ry)
             (vx' = 0) (vy' = 0));
        (and (rx < 3) (ry >= 3)) =>
        (and (ix' = true) (iy' = false)
             (rx' = rx) (ry' = ry)
             (vx' = 0) (vy' = 0));
}
#  car1 : dec, dec
{ mode: ix = false; iy = false;
  inv:  rx > -20; ry > -20;
        vx > -8; vy > -8;
  flow: d/dt[rx] = vx;
        d/dt[ry] = vy;
        d/dt[vx] = -1.2;
        d/dt[vy] = -1.4;
  jump: (and (rx < 3) (ry < 3)) =>
        (and (ix' = true) (iy' = true)
             (rx' = rx) (ry' = ry)
             (vx' = 0) (vy' = 0));
        (and (rx >= 3) (ry < 3)) =>
        (and (ix' = false) (iy' = true)
             (rx' = rx) (ry' = ry)
             (vx' = 0) (vy' = 0));
        (and (rx < 3) (ry >= 3)) =>
        (and (ix' = true) (iy' = false)
             (rx' = rx) (ry' = ry)
             (vx' = 0) (vy' = 0));
}
init: not(ix); 0 <= rx; rx <= 1;
      not(iy); 0 <= ry; ry <= 1;
      -2.5 <= vx; vx <= -2;
      -2.5 <= vy; vy <= -2;

proposition:
  [pix]: ix;

# bound: 10, timebound: 10, solver: yices
goal:
  [f1]: [][0,4] (vx < -2 -> <>[2,5] rx <= -2);
  [f2]: (<>[0, 4] ry > 3) U [0, 5] (vy > 1.5);
  [f3]: <>[0,3] (rx <= 5 U[0, 5] pix = false);
```

Fig. 12: An input model of the autonomous car

A.4 A Railroad Gate Controller

Figure 13 is a model file of the railroad gate controller in STLMC.

```
bool a;    bool b;
[-20, 100] x; [0, 90] pb; [-50, 50] vb;
# far
{ mode: a = false; b = false;
  inv:  x > 60;
  flow: d/dt[x] = -30;
        d/dt[pb] = vb;
        d/dt[vb] = 0;
  jump: x < 80 => (and (a' = a) (b' = true)
                       (pb' = pb) (x' = x)
                       (vb' = 6));
        x < 70 => (and (a' = true) (b' = b)
                       (pb' = pb)
                       (x' = x) (vb' = 8));
}
# approach
{ mode: a = false; b = true;
  inv:  x > 20;
  flow: d/dt[x] = -5;
        d/dt[pb] = vb;
        d/dt[vb] = 0.3;
  jump: x < 80 => (and (a' = true) (b' = false)
                       (pb' = pb)
                       (x' = x) (vb' = vb));
}
# close
{ mode: a = true; b = false;
  inv:  x > 10;
  flow: d/dt[x] = -5;
        d/dt[pb] = vb;
        d/dt[vb] = 0.5;
  jump: x < 20 => (and (a' = a) (b' = true)
                       (pb' = pb)
                       (x' = x) (vb' = -4));
}
# past
{ mode: a = true; b = true;
  inv:  x > -10;
  flow: d/dt[x] = -5;
        d/dt[pb] = vb;
        d/dt[vb] = -1;
  jump: (x < 0) => (and (a' = false)
                        (b' = false)
                        (pb' = pb) (vb' = 0)
                        (x' = 100 + x));
}
init: a = false; 89 <= x; x <= 90;
      b = false; 0 <= pb; pb <= 0.5;
      vb = 0;
proposition:

# bound: 10, timebound 20, solver: yices
goal:
  [f1]: <>[0, 5] ((pb >= 40) U[1, 8] (x < 40));
  [f2]: <>[0, 4](x < 50 -> [][2,10] pb > 40);
  [f3]: [][0.0,5.0) ((x < 50) U[2, 10] (pb > 5));
```

Fig. 13: An input model of the railroad gate controller

A.5 A Filtered Oscillator

Figure 14 is a model file of the filtered oscillator in STLmc.

```
real loc;
[-100, 100] x;  [-100, 100] y;  [-100, 100] z;
[-100, 100] x1; [-100, 100] x2; [-100, 100] x3;
{ mode: loc = 1;
  inv:  x >= 0;
        y + 2 * x >= 0;
  flow: d/dt[x] = -0.2 * x + 1.4;
        d/dt[y] = -0.1 * y - 0.7;
        d/dt[x1] = 3 * x - 3 * x1;
        d/dt[x2] = 3 * x1 - 3 * x2;
        d/dt[x3] = 3 * x2 - 3 * x3;
        d/dt[z] = 3 * x3 - 3 * z;
  jump: (and (x >= 0) (y + 2 * x <= 0)) =>
        (and (loc' = 2) (x' = x) (y' = y)
             (x1' = x1) (x2' = x2) (x3' = x3)
             (z' = z));
}
{ mode: loc = 2;
  inv:  x >= 0;
        y + 2 * x <= 0;
  flow: d/dt[x] = -0.2 * x - 1.4;
        d/dt[y] = -0.1 * y + 0.7;
        d/dt[x1] = 3 * x - 3 * x1;
        d/dt[x2] = 3 * x1 - 3 * x2;
        d/dt[x3] = 3 * x2 - 3 * x3;
        d/dt[z] = 3 * x3 - 3 * z;
  jump: (and (x <= 0) (y + 2 * x <= 0)) =>
        (and (loc' = 3) (x' = x) (y' = y)
             (x1' = x1) (x2' = x2) (x3' = x3)
             (z' = z));
}
{ mode: loc = 3;
  inv:  x <= 0;
        y + 2 * x <= 0;
  flow: d/dt[x] = -0.2 * x - 1.4;
        d/dt[y] = -0.1 * y + 0.7;
        d/dt[x1] = 3 * x - 3 * x1;
        d/dt[x2] = 3 * x1 - 3 * x2;
        d/dt[x3] = 3 * x2 - 3 * x3;
        d/dt[z] = 3 * x3 - 3 * z;
  jump: (and (x <= 0) (y + 2 * x >= 0)) =>
        (and (loc' = 4) (x' = x) (y' = y)
             (x1' = x1) (x2' = x2) (x3' = x3)
             (z' = z));
}
{ mode: loc = 4;
  inv:  x <= 0;
        y + 2 * x >= 0;
  flow: d/dt[x] = -0.2 * x + 1.4;
        d/dt[y] = -0.1 * y - 0.7;
        d/dt[x1] = 3 * x - 3 * x1;
        d/dt[x2] = 3 * x1 - 3 * x2;
        d/dt[x3] = 3 * x2 - 3 * x3;
        d/dt[z] = 3 * x3 - 3 * z;
  jump: (and (x >= 0) (y + 2 * x >= 0)) =>
        (and (loc' = 1) (x' = x) (y' = y)
             (x1' = x1) (x2' = x2) (x3' = x3)
             (z' = z));
}
init: loc = 1; 0.2 <= x; x <= 0.3;
               -0.1 <= y; y <= 0.1;
      x1 = 0; x2 = 0; x3 = 0; z = 0;


proposition:

# bound: 5, timebound: 8, solver: dreal
goal:
  [f1]: <>[0,3]((x3 >= 1) R[0, inf) (y <= 10));
  [f2]: <>[2, 5] ([][0, 3] (x2 < 4));
  [f3]: ([][1, 3] (x <= 2)) R[2, 5] (x3 > 2);
```

Fig. 14: An input model of a filtered oscillator